CS209 Final Project Report

Student ID	Name	Contribution	Main Work
12012640	黄悦童	50%	Build a front-end interactive platform, Data visualization and analysis
12011412	杨鹭鸣	50%	Data crawling and processing, Restful Apis Implementation

Work done

Basis

- [x] Use Java to craw data from Stack Overflow
- [x] Process data on 'Answers', 'Accepted Answer', 'Tags', 'Users'
- [x] Visualize graphs properly
- [x] More than 500 threads calculated
- [x] Use Spring Boot as backend frame and Vue.js as frontend frame

Bonus

- [x] Craw data on 'Frequently Discussed Apis'
- [x] Set up >3 restful Apis returning processed data

Data crawling

`RunQandA.java` and `RunComment.java`

`RunQandA.java` is a program that gets questions and answers from the Stack Overflow API. It retrieves a list of questions related to Java programming language from the Stack Overflow website, along with their corresponding answers, and performs various analyses on the data.

`RunComment.java` is similar to runQandA, it gets comments from Stack Overflow API.

In order to reduce the number of visits, we grouped 100 pieces of data, packaged the application and queried it with the server. This can significantly reduce the number of times the StackOverflow API is used.

Libraries

1. `okhttp3`: It is an open-source library for making HTTP requests in Java. In this code, it is used to send HTTP requests to the Stack Overflow API and receive responses. It provides a convenient and efficient way to handle network communication.

2. `org.json`: It is a library for processing JSON data in Java. It provides classes and methods to parse JSON strings, create JSON objects and arrays, and manipulate JSON data. In this code, it is used to parse the responses received from the Stack Overflow API, extract relevant information, and store the data in JSON format.

functionality:

Importing Libraries:

The required libraries are imported, including org.json for JSON processing and okhttp3 for making HTTP requests.

API_KEY Constant:

The API key for accessing the Stack Overflow API is defined as a constant variable.

main Method:

The main method serves as the entry point of the program. It calls the analyzeQuestions method to start the analysis.

analyzeQuestions Method:

This method performs the main analysis on the Stack Overflow questions and answers.
It initializes variables to store various metrics and collections for questions and answers.
A loop is used to iterate over multiple pages of questions (up to a maximum of 10 pages) from the API.
For each page of questions, the getQuestions method is called to retrieve the questions.
The method then processes each question, retrieves its answers using the getAnswers method, and collects the necessary data.
Metrics such as the total number of questions, questions without answers, total number of answers, and maximum number of answers are calculated.
The collected question and answer data is stored in separate collections.
Finally, the metrics are printed, and the question and answer data are written to local JSON files.

getQuestions Method:

This method constructs the API URL and sends an HTTP request to retrieve a page of questions from the Stack Overflow API.
The URL is constructed with parameters specifying the sorting order, site (Stack Overflow), API key, page number, page size, date range, and tags (only Java-related questions).
The method uses the sendRequest method to execute the HTTP request and parse the response into a JSON object, which is returned.

getAnswers Method:

Similar to the getQuestions method, this method constructs the API URL and sends an HTTP request to retrieve answers for a specific question ID from the Stack Overflow API.
The URL is constructed with parameters specifying the question ID, sorting order, site (Stack Overflow), and API key.
The method uses the sendRequest method to execute the HTTP request and parse the response into a JSON object, which is returned.

sendRequest Method:

This method uses the okhttp3 library to send an HTTP GET request to the specified URL and retrieve the response.
It returns the response as a JSON object after parsing the response body.

getAnswerList Method:

This utility method converts a JSON array of answer items into a list of JSONObject objects for easier processing.

writeJSONToFile Method:

This utility method writes a list of JSONObject data to a JSON file with the specified filename.
It uses a PrintWriter to write the JSON data as a string to the file.

Data Processing

`contextAnalyze.java` and `generateResult.java`

In data analysis, we analyze three sets of data packets in various ways, namely the ones mentioned earlier: `questions.json` `answers.json` and `comments.json`.

In `generateResult.java`, we analyze the data from stackoverflow API and save the result to another json file called `result.json` . So that back end RESTful API can get data from it and prevent analyzing step in backend program.

The code performs various analyses on data stored in JSON files and generates a final result in the form of a dictionary, which is then saved in a file named "result.json".

The code is divided into several functions, each responsible for a specific task:

1. `find_answer_info(question_id)` : This function takes a question ID as input and searches for answers related to that question in the "answers.json" file. It counts the number of answers, checks if there is an accepted answer, records the solution time, and compares the scores of the answers to determine if any answer has a higher score than the accepted answer.
2. `generate()` : This function calculates various metrics based on the data in the "questions.json" file. It iterates through the questions, calls `find_answer_info()` for each question, and collects information such as the total number of questions, questions without answers, average number of answers per question, maximum number of answers to a question, distribution of answer counts, percentage of questions with accepted answers, percentage of answers with better scores than the accepted answer, and resolution time distribution.
3. `analyze_tags()` : This function analyzes the tags associated with the questions in the "questions.json" file. It calculates tag co-occurrence counts, upvotes for each tag or tag combination, and views for each tag or tag combination. It then sorts and selects the most frequent tag co-occurrences, tags with the highest upvotes, and tags with the highest views.
4. `count_distinct_users(data)` : This function counts the number of distinct users based on the "user_id" field in the provided data.

5. `get_most_active_users(data, num_users)` : This function identifies the most active users based on their contributions in the provided data. It counts the occurrences of each user and returns a list of the top "num_users" users sorted by their activity count.
6. `get_user_info(data, user_id)` : This function retrieves additional information about a specific user based on their "user_id" in the provided data.

The main part of the code starts by calling the `generate()` and `analyze_tags()` functions to collect the metrics and tag analysis results, respectively. It then proceeds to perform user-related analyses:

1. It initializes variables to track the count of non-existent user IDs and the total user count.
2. It loads the data from the "answers.json", "questions.json", and "comments.json" files into separate variables.
3. It calls the `count_distinct_users()` function for each type of data to calculate the number of distinct users who posted questions, answers, and comments.
4. It prints the counts of distinct users and posts without user IDs.
5. It calls the `get_most_active_users()` function to retrieve the most active users in the discussions and prints their user IDs and activity counts.
6. It calls the `reputation_score_relation()` and `reputation_acceptance_relation()` functions to analyze the relationship between user reputation and post score, and between user reputation and answer acceptance, respectively.
7. It saves the final result dictionary into a JSON file named "result.json".

In `contextAnalyze.java`, we analyze the data of body part of posts in order to know what API is discussed most in stackOverflow.

It performs context analysis on data extracted from the Stack Overflow website. It collects data from JSON files containing information about questions, answers, and comments from Stack Overflow. The code then processes the data to identify frequently discussed Java classes and methods within the collected content.

The contextAnalyze class: contains the main method that serves as the entry point of the program.

Various JSON arrays (questions, answers, comments) are declared to store the extracted data from corresponding JSON files.

The main method: loading the JSON data from the files using the loadJSONArrayFromFile method. Then extracts the IDs from the JSON arrays using the extractIds method.

Two separate lists (allIds, commentIds) are created to store all the question and answer IDs together and all the comment IDs, respectively.

The getPostContent method: retrieve the content (body) of posts (questions and answers) using the obtained IDs. The method makes API requests to the Stack Exchange API to retrieve the data in batches. The retrieved post content is stored in a list of JSON objects (allPosts).

The getCommentContent method: retrieve the content (body) of comments using the obtained comment IDs. Similar to getPostContent, this method also makes API requests in batches to retrieve the data. The retrieved comment content is stored in a list of JSON objects (allComments).

The extracted bodies from posts and comments are combined into a single list (allBodies), and the list is saved as a JSON array in a file named "all_bodies.json". Then proceeds to extract Java API mentions from the collected bodies using the extractJavaAPIs method.

It searches for combinations of capitalized words followed by a dot, indicating a potential class and method reference. The identified combinations are stored in two separate lists (apiMentionsClasses, apiMentionsMethods).

A JSON object (result) is created to store the frequent classes and methods.

The libraries/packages:
java.nio.charset.StandardCharsets for specifying character encoding.
org.json for working with JSON data.

RESTful APIs Documentation

Obtain `result.json` file.

```
    @GetMapping("/result")
    public ResponseEntity<byte[]> getFile() throws IOException
```

Single number APIs including int, $double$. Check corresponding function name.

```
    @GetMapping("/PercentageWithoutAnswers")
    public ResponseEntity<Double> getPercentageWithoutAnswers() throws IOException
```

```
    @GetMapping("/AverageAnswers")
    public ResponseEntity<Double> getAverageAnswers() throws IOException
```

```
    @GetMapping("/MaxAnswers")
    public ResponseEntity<Integer> getMaxAnswers() throws IOException
```

Return JsonNode of answer distribution.

```
    @GetMapping("/AnswerDistribution")
    public ResponseEntity<JsonNode> getAnswerDistribution() throws IOException
```

```
    @GetMapping("/AcceptedAnswersPercentage")
    public ResponseEntity<Double> getAcceptedAnswersPercentage()
```

Return a list of integer including resolution time distribution.

```
    @GetMapping("/QuestionResolutionTimeDistribution")
    public ResponseEntity<List<Integer>> getQuestionResolutionTimeDistribution()
```

```
    @GetMapping("/NonAcceptedAnswersUpvotesPercentage")
    public ResponseEntity<Double> getNonAcceptedAnswersUpvotesPercentage()
```

Return List of Map, which mapping Tag names to occur count.

```
    @GetMapping("/FrequentlyCooccurringTags")
    public ResponseEntity<List<Map<String, Object>>> getFrequentlyCooccurringTags()
```

Return List of Map, which mapping Tag names (and tag combination name) to upvote count.

```
    @GetMapping("/MostUpvotedTags")
    public ResponseEntity<List<Map<String, Object>>> getMostUpvotedTags()
```

Return List of Map, which mapping Tag names (and tag combination name) to view count.

```
    @GetMapping("/MostViewedTags")
    public ResponseEntity<List<Map<String, Object>>> getMostViewedTags()
```

```
    @GetMapping("/QuestionUserNumber") //参与问题的
    public ResponseEntity<Integer> getQuestionUserDistribution()
```

```
    @GetMapping("/AnswerUserNumber")
    public ResponseEntity<Integer> getAnswerUserNumber()
```

```
    @GetMapping("/CommentUserNumber")
    public ResponseEntity<Integer> getCommentUserNumber()
```

Return List of Tuple (List of 2), One of number is unique user participating in answer, the other is in comment.
Param available, `num=x` means return first x data.

Usage: `http://localhost:9090/SingleQuestionUserDistribution?num=5`

```
    @GetMapping("/SingleQuestionUserDistribution")
    public ResponseEntity<List<List<Integer>>> getSingleQuestionUserDistribution(
            @RequestParam(value = "num", required = false, defaultValue = "1000")
int num
    )
```

```
    @GetMapping("/NumberOfPostsWithoutUserID")
    public ResponseEntity<Integer> getNumberOfPostsWithoutUserID()
```

```
    @GetMapping("/TotalNumberOfUsers")
    public ResponseEntity<Integer> getTotalNumberOfUsers()
```

```
    @GetMapping("/MostActiveUsers")
    public ResponseEntity<List<JsonNode>> getMostActiveUsers() throws IOException
```

```
    @GetMapping("/FrequentlyDiscussedClasses")
    public ResponseEntity<Map<String, Integer>> getFrequentlyDiscussedClasses()
```

```
    @GetMapping("/FrequentlyDiscussedMethods")
    public ResponseEntity<Map<List<String>, Integer>>
getFrequentlyDiscussedMethods()
```

```
    @GetMapping("/ReputationScoreRelation")
    public ResponseEntity<JsonNode> getReputationScoreRelation()
```

```
    @GetMapping("/ReputationAcceptanceRelation")
    public ResponseEntity<JsonNode> getReputationAcceptanceRelation()
```

RESTful API to get user's total score in posts in database.

Usage: `http://localhost:9090/UserTotalScore/5316516`

```
    @GetMapping("/UserTotalScore/{UserId}")
    public ResponseEntity<Integer> getUserTotalScore(@PathVariable("UserId") int
userId)
```

Data storage pattern

Number of Answers

What percentage of questions don't have any answer?

```
final_result['percentage_without_answers']
```

What is the average and maximum number of answers?

```
final_result['average_answers']
final_result['max_answers']
```

What is the distribution of the number of answers?

The distribution contains answer count of:

0, 1, 2, 3, 4, 5, 6, 7+

```
answer_distribution = {
        "0",
        "1",
        "2",
        "3",
        "4",
        "5",
        "6",
        "7+"
    }
final_result['answer_distribution']
```

Accepted Answers

What percentage of questions have accepted answers (one question could only have one accepted answer)?

```
final_result['percentage_have_accepted_answer']
```

What is the distribution of question resolution time (i.e., the duration between the question posting time and the posting time of the accepted answer)?

```
final_result['percentage_better_than_accepted']
```

What percentage of questions have non-accepted answers (i.e., answers that are not marked as accepted) that have received more upvotes than the accepted answers?

```
final_result['resolution_time_distribution']
```

Tags

Which tags frequently appear together with the java tag?

```
results['sorted_cooccurrence']:[other_tag, count]
```

Which tags or tag combinations receive the most upvotes?

```
results['sorted_upvotes']:[[tags or combinations], upvotes]
```

Which tags or tag combinations receive the most views?

```
results['sorted_views']:[[tags or combinations], views]
```

Users

Many users could participate in a thread discussion. What is the distribution of such participation (i.e., the number of distinct users who post the question, answers, or comments in a thread)?

```
final_result['number_of_question_users']
final_result['number_of_answer_users']
final_result['number_of_comment_users']
```

Addition: Please mention, number of posts without user id, A pie chart can be made to indicate that some people have no ID or have cancelled their account.

```
final_result['single_question_user_distribution']:list(tuple(x,y))
```

A thread has x responses and y comments, and the distribution of these two numbers is calculated through 800 threads

```
final_result['number_of_posts_without_user_id']

final_result['total_number_of_users'
```

Which are the most active users who frequently participate in thread discussions?

```
final_result['most_active_users']:[user_id:['info', 'count']]
```

Frequently discussed Java APIs

Which Java APIs (e.g., classes, methods) are frequently discussed on Stack Overflow?

```
result["frequent_classes"] = class_result[{"name", "count"}]
result["frequent_methods"] = method_result[{"method", "class", "count"}]
```

Addition

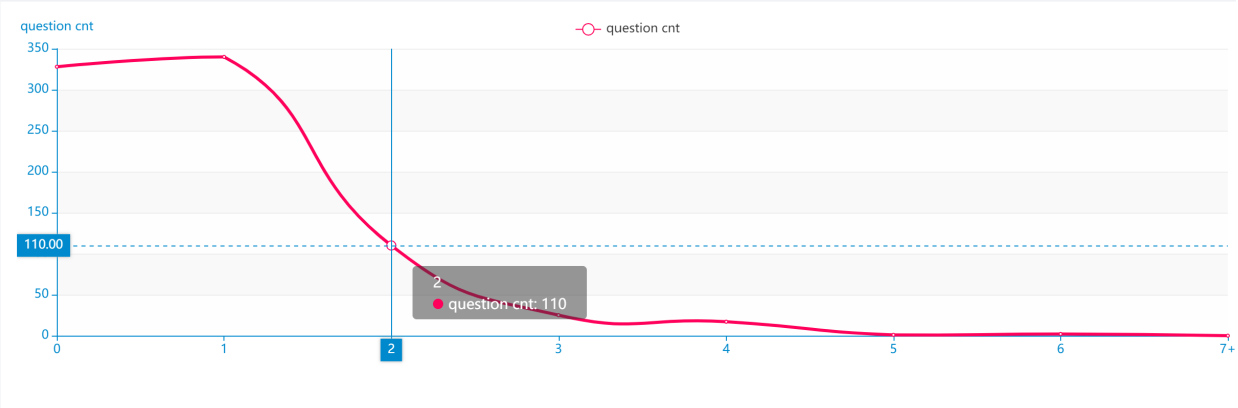
```
final_result['reputation_score_relation'] = [user_id: [reputation, score]]
final_result['reputation_acceptance_relation'] = [user_id: [reputation,
is_accepted]]
```

Can use this data to create a scatter plot and see if there is a relationship between reputation and score/acceptance in the results

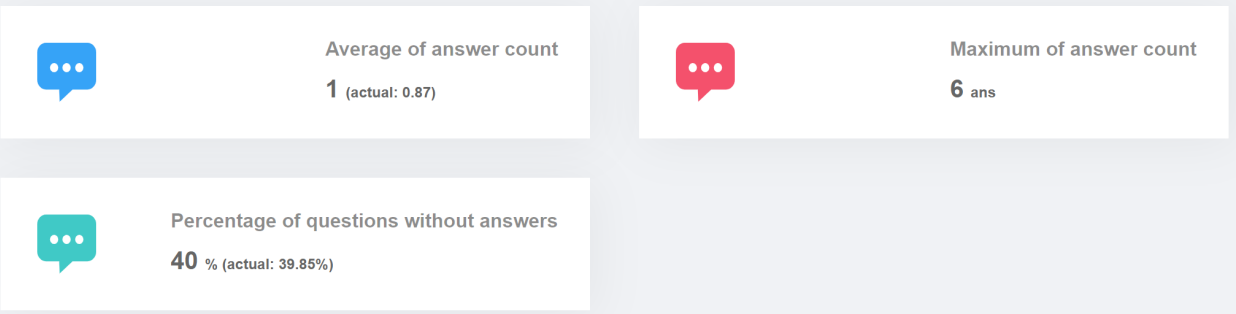
Visualization and Insights

Number of Answers

- Line chart of number of questions with answers from 0 to 7+:



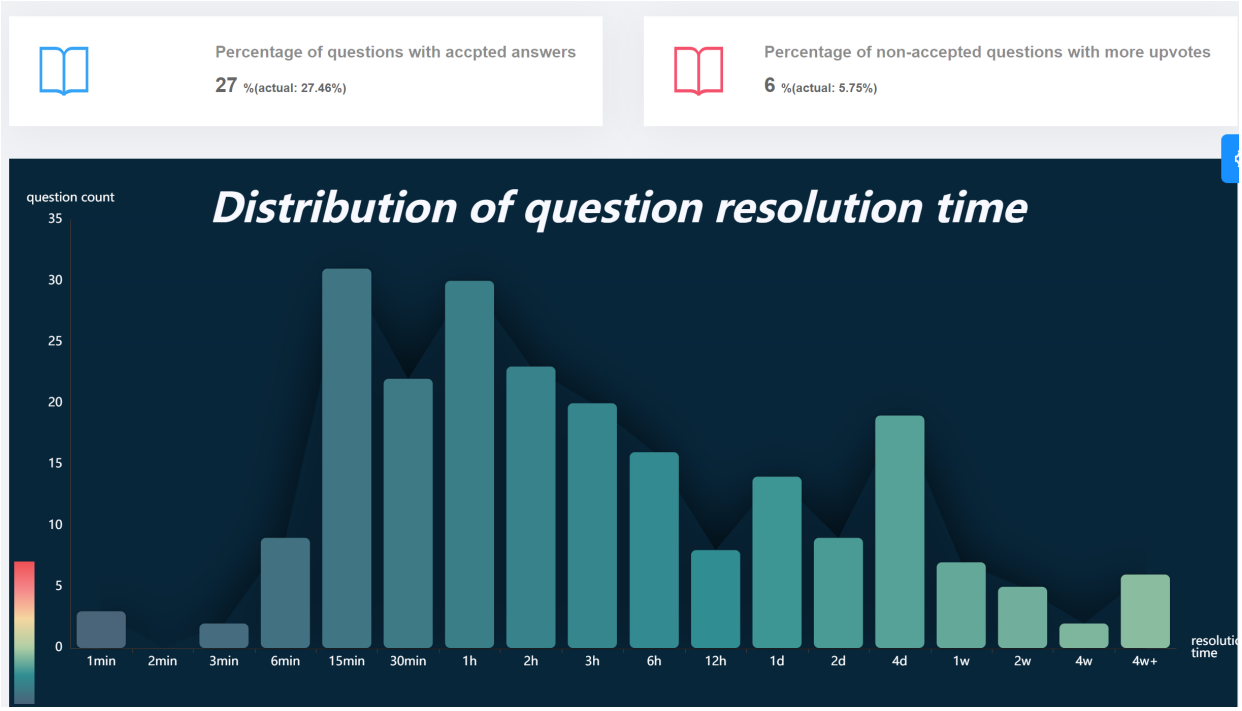
- Data:



- Insights
  - Most questions have no answer or only one answer.
  - The questions with answers equal to or more than 2 are all above 110.
  - The percentage of questions without answers is 40%, which means that there is a large possibility that a question will be ignored by the users among society.
  - Most questions will only get one specific answer, and only a small proportion(below 1%) of questions will luckily have 6 answers.

Accepted Answers

- Data and distribution of question resolution time:

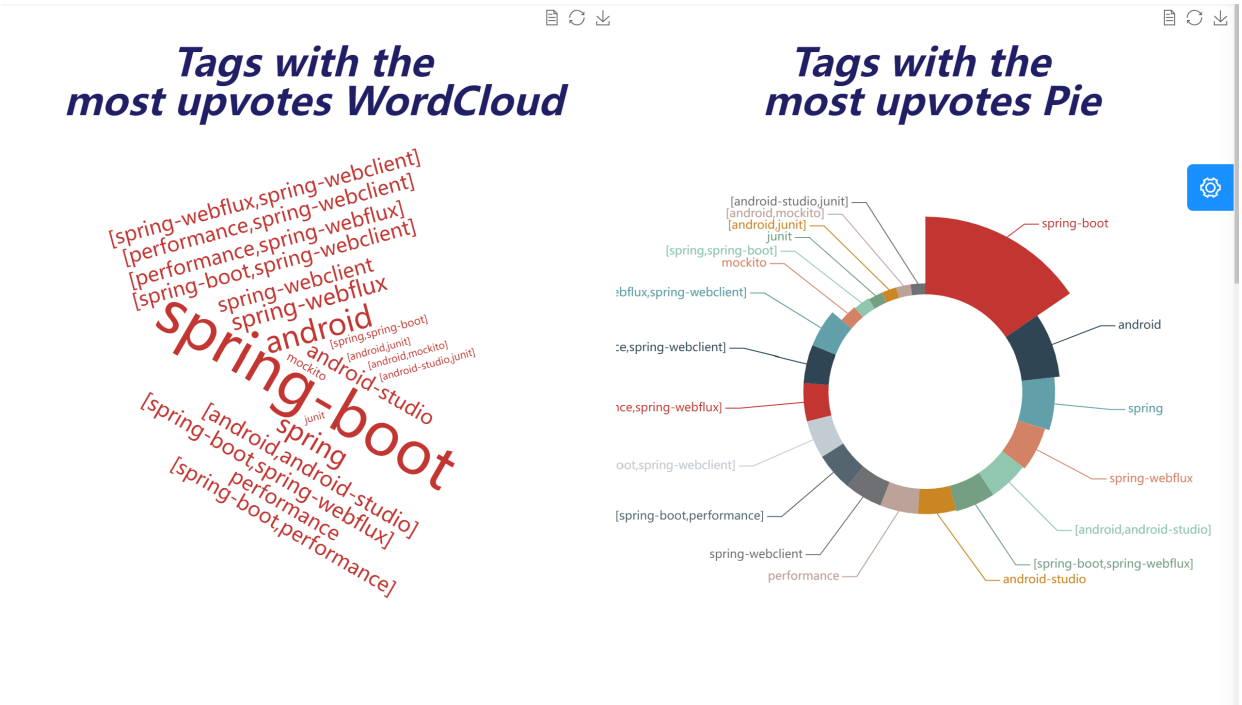


- Insights
  - We have already known there is approximately 60% of questions will be answered. Among them, only 27% will have accepted answers. Therefore, about 18% of questions will have accepted answered, which means they are well solved.
  - Nearly 80% of questions have no answer or their answers are not excellent enough for problem solving,
  - More than 70% questions will have accepted answer in the time duration of [15min, 12h], which is a pretty decent pace for getting answer.
  - Nearly 20% of questions can't get answer they want on the day they are posted. It is very likely that Stack Overflow needs new strategies to attract people to engage in question solving more.
  - Only 6% of non-accepted questions will have more upvotes, so most of them might also be less popular topics.

Tags

Tags with most upvotes

- Data



Single tags among top 20

tag	value
spring-boot	71
android	35
spring	30
spring-webflux	26
android-studio	23
performance	23
spring-webclient	23
mockito	12
junit	10

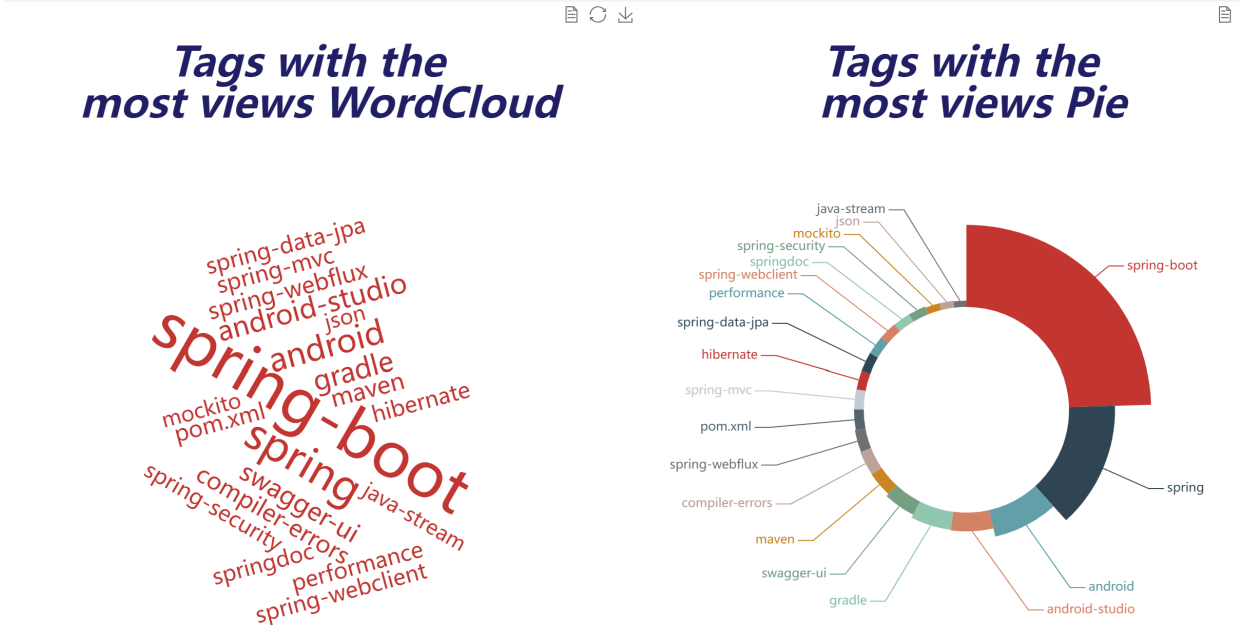
Combination tags among top 20

tag1	tag2	value
android	android-studio	24
spring-boot	spring-webflux	24
spring-boot	performance	23
spring-boot	spring-webclient	23
performance	spring-webflux	23
performance	spring-webclient	23

- Insights
  - Single tags and Combination tags among the top 20 are half and half.
  - Spring Boot is the top of all and the most popular tag.
  - Among single tags, people care about Spring Boot and Android the most, and Spring also caught some attention
  - Users tend to search combination between Android and Spring Boot, which means there might be comparison or aggregation of functions and characteristics of them.

Tags with most views

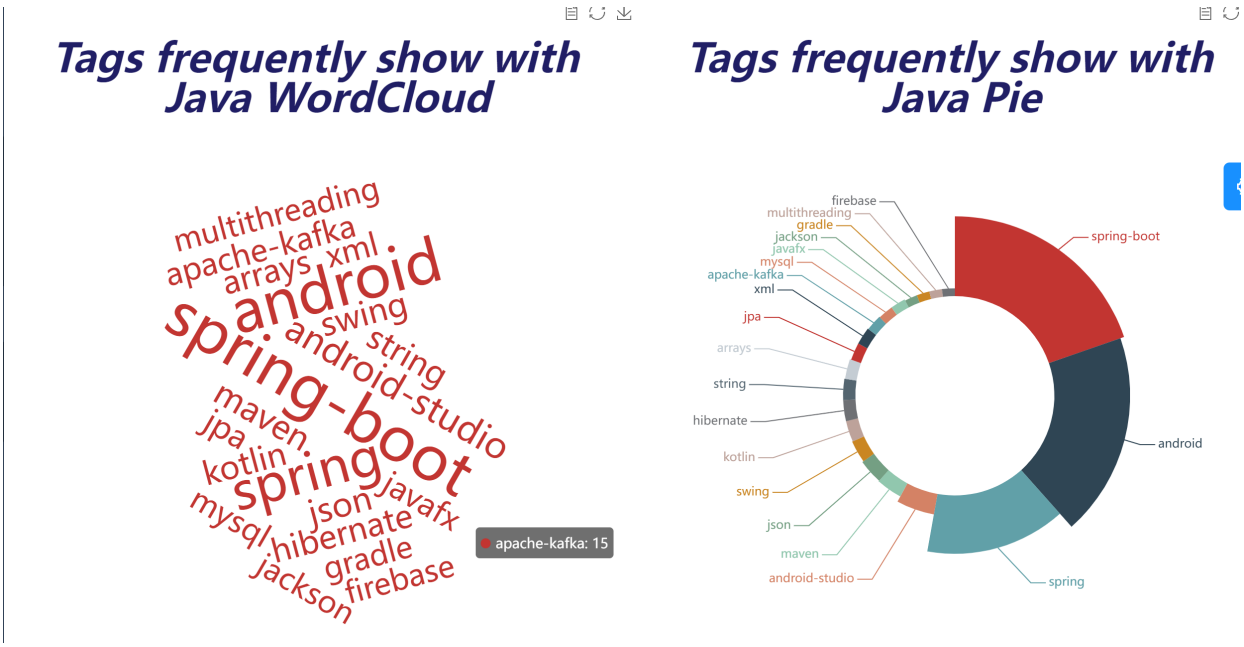
- Data



- Insights
  - spring-boot, spring, and android are the top3 tags with the most views
  - No combination tags are most viewed which means that most users tend to look for specific information in one domain
  - Some plugins and useful tool of Spring Boot are also widely viewed, such as maven, swagger-ui.

Tags frequently show with Java

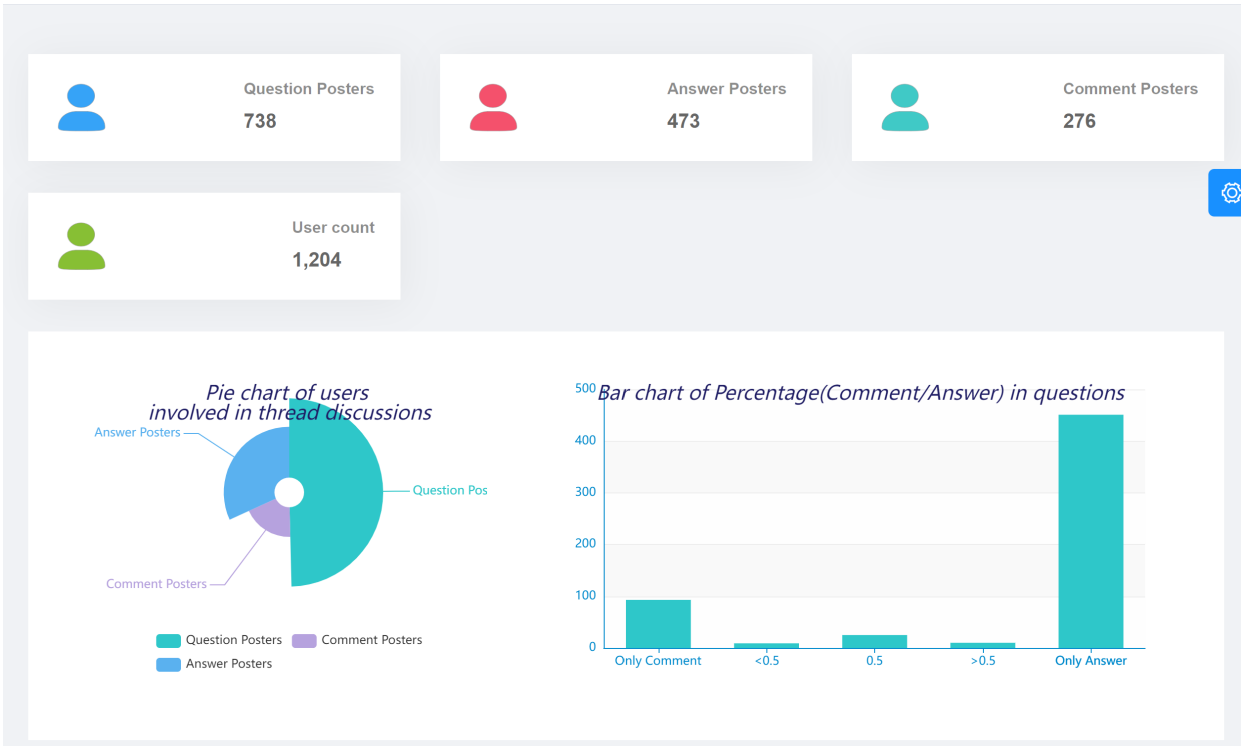
- Data



- Insights
  - spring-boot, android, spring has the highest exposure frequency with Java, showing high popularity as a framework.
  - kotlin is in the top 10. As a new language, Kotlin is compatible with Java, which means you can use Java code and libraries in Kotlin. It starts to have a higher profile.

Users

- Data



Active Users in Threads

Top 10 active users

☑ name ☐ id ☑ reputation ☐ link ☑ participation

name	reputation	participation
Queeg	7325	15
rzwitserloot	81301	14
Alex Mamo	127416	14
Michail Alexakis	1367	13
ganjaam	933	11
VanessaF	544	10
Darkman	2906	9
Alexander Ivanchenko	24802	9
Holger	282655	9
TSLee	23	8

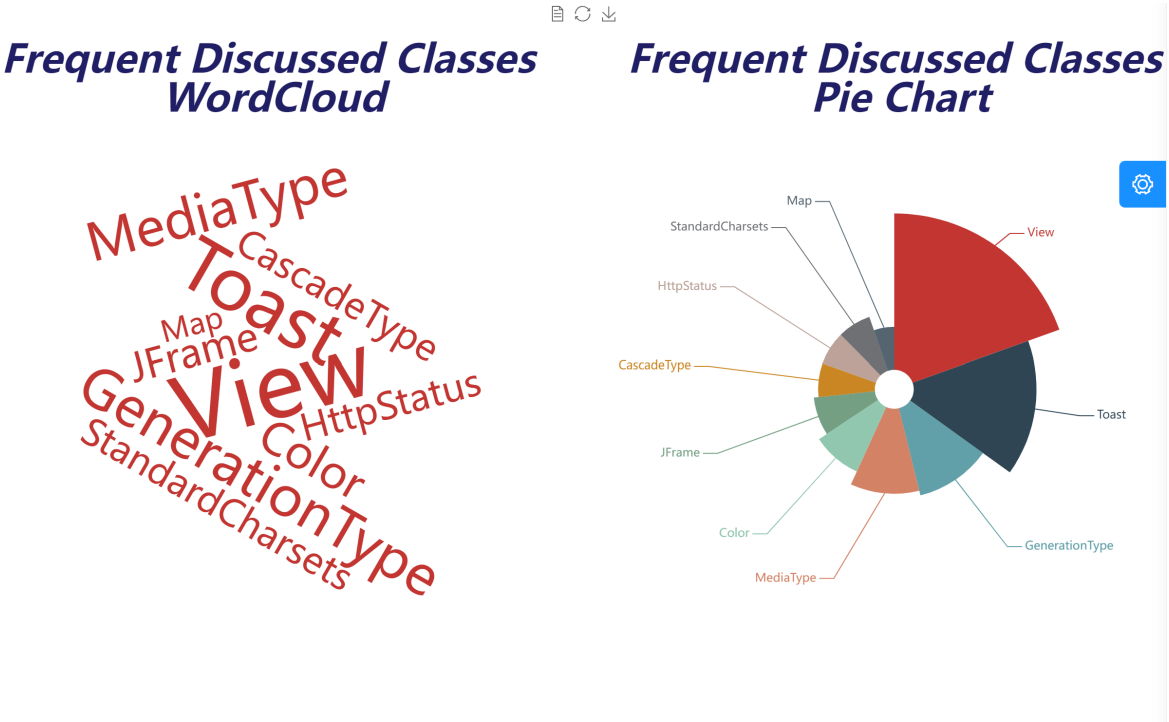
- Insights

- Queeg, rzwitserloot, Alex Mamo participated the most in the questions solving, and they all have high reputation over 5000.
- The number of question posters are nearly 45% larger than that of answer posters and nearly 200% larger than that of comment posters.
- Five-sixth of questions only have answers, while one-sixth of questions only have comments.
- Problem solving is a tendency and people tends to comment less.

Frequently discussed Java APIs

Frequently discussed classes

- Data



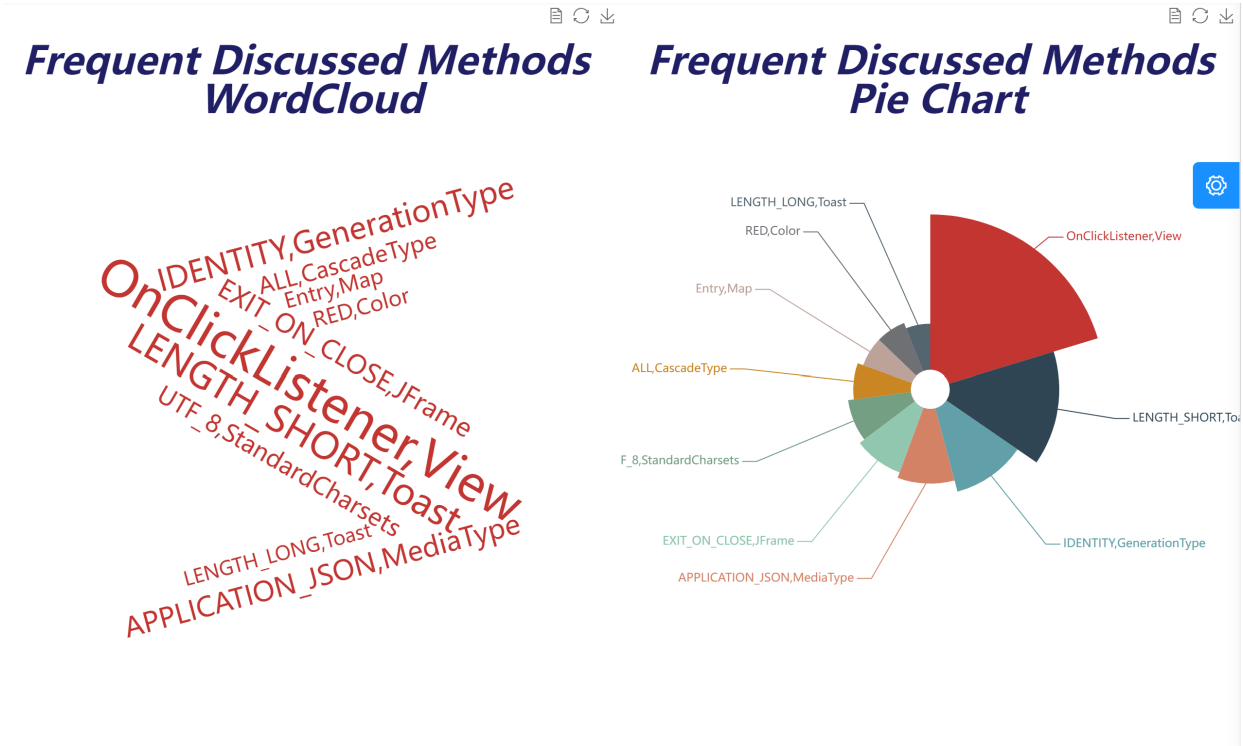
name	value
View	33
Toast	26
GenerationType	19
MediaType	18
Color	15
JFrame	13
CascadeType	12
HttpStatus	12
StandardCharsets	12
Map	9

- Insights

- View and Toast are very common classes in Android application development, so they appear more frequently in a given piece of data. Because of the large number of user interface elements required to develop Android applications, the View class is used more frequently in code. Developers often use the Toast class to display short prompt messages in their applications, so it appears relatively frequently in code.
- The Android platform is currently one of the most widely used mobile operating systems in the world, so developing Android apps is in high demand, and the View and Toast classes are an important part of building the Android app interface and providing user feedback, so they are likely to be popular classes.

Frequently discussed methods

- Data



method	class	value
OnClickListener	View	27
LENGTH_SHORT	Toast	19
IDENTITY	GenerationType	15
APPLICATION_JSON	MediaType	13
EXIT_ON_CLOSE	JFrame	12
UTF_8	StandardCharsets	11
ALL	CascadeType	10
Entry	Map	9
RED	Color	9
LENGTH_LONG	Toast	8

- Insights

- OnClickListener for View: A common way for users to interact with the interface in Android applications is by adding an OnClickListener to the View. OnClickListener is an interface that defines the actions to be performed when the user clicks a View. Developers use OnClickListener to implement interactions such as button clicks, list item selections, and so on. Because user interaction is a core part of Android application development, View's OnClickListener is used more frequently in code.
- Toast's LENGTH_SHORT: Toast is a lightweight message notification mechanism for displaying short prompts or notifications to the user. LENGTH_SHORT is the length of the Toast display, indicating that the message will be displayed on the screen for a shorter amount of time. Developers often use LENGTH_SHORT to ensure that prompt messages can be displayed quickly and do not take up too much time and space in the user interface. Since Toast is commonly used by developers to provide short term feedback to users, LENGTH_SHORT has become a common choice.